ATLAS Trigger & DAQ

The raw event format in the ATLAS Trigger & DAQ

Document version/issue:	4.0c
Document date:	09 February 2009
Document status:	**Final**
Document Reference:	ATL-D-ES-0019

Abstract

This note presents the ATLAS raw event format. It covers the format of data from the ReadOut Drivers to the output of the Event Filter. It does not cover the detector specific event data.

Editors:

André dos Anjos – University of Wisconsin/Madison

Hans Peter Beck – University of Bern

Benedetto Gorini – CERN

Issue	Revision	Date	Reason for change
1	0	01 Apr ‘97	Birth.
1	1	07 Oct. ‘97	General update of all sections.
1	2	20 Oct. ‘97	Muon sub-detector IDs changed at the request of S. Falciano.
1	3	14-Aug. ‘98	Add an offset to ROD trailer that indicates the relative order of Data/status information. General clean-up. Appendix with an initial header file and an appendix of an example use of the header file.
1	4	05-Sept.’98	Redefined last word of ROD trailer. Comments on sub-detector ID’s from Philippe Farthouat. Comments from Jorgen Petersen.
1	5	15-Oct. ‘98	Tidy-up ready for release as ATLAS note Remove Appendices.
2	0	11-Mar.‘02	Include feedback on version 1.5 from detector community Increase scope to include Level 2; Change from DAQ -1 specific terminology; Re-define Source ID; Change Level 1 ID element to be defined as the combination of the 24-bit L1ID and the 8-bit ECRID; Add mechanism to determine byte ordering dynamically; Remove section on ROL implementation; Change unit of Total fragment size and Header size element to be 32-bit integer; Re-define the Format version number so that it may also be used to identify the format version of the detector Data; Distribute to author list; General Distribution
2	2	11-Oct. ‘02	Added Appendix A on ROL implementation issues; Clean-up of section 2.1 (main requirements); Implementation of Source ID element re-defined, i.e. Module ID now byte wide, see section 5.2; Global event ID removed from ROS specific header, section 5.10.3; Introductory text in section 5 re-written; Description of “Format version number” element re-written, section 5.6; Corrected description of Bunch Crossing ID, Tables 8, 9 & 10; Table 3 defining values for Sub-detector IDs updated to match known TTC partitions; Error in description of extended level 1 ID corrected, sections 4 & 5.10.1; Clarified meaning of Detector Event Type element, section 4; Remove Level 1 Trigger Info. from Full Event Specific header, section 5.10.1; Initial values and meanings for generic status field (adopted from Level1 - Data-Flow interface document), section 5.8; Remove LVL2-Data and LVL2-Result, table 2; Added RoI Builder Module Type, table 4; Deleted sections 5.10.6 and 5.10.7; Expanded scope to include output of Event Filter, section 1.3; Added section on Event Filter Output, section 5.11.

2	4	1 Feb. ‘04	Date and time element in Full Event Specific element (Section 5.10.1) redefined to be the number of seconds elapsed since 00h00.00 on 1st Jan. 1970, i.e. in line with Posix; Run number added to Generic fragment and ROD fragment (section 3,4 and 5). Remove run number from Full Event Specific element (section 5.9.1) and ROS Specific Header (section 5.9.3); A default sub-detector type added to Table 3. to be used for equipment which is not specific to any single detector. Module type ‘Level 2 Processor’ changed to ‘HLT Processor’; Added ‘Event Filter Info’ in section 5.9.1 and Table 6; Changed ‘Level 1 ID’ to ‘Extended Level 1 ID’ where appropriate; Updated some of the references; Cleaned up some typing errors.
		23 Feb. ‘04	Added Appendix B which details additional values of the Sub-detector IDs to be used in the 2004 Combined test beam; Changed the order of Extended Level 1 ID and Bunch Crossing ID in table 9 so as to be the same as in tables 10 and 11.
2	5	11 Mar. ‘05	Dropped Module Type as same functionality achieved with a combination of Header marker and Sub-detector Ids; Additional TDAQ Sub-detector IDs, page 13, to account for removal of Module Type Source ID re-defined, page 11, to allow more than 256 module Ids; Removed Offset elements from generic header. Redundant infor- mation and re-duces overall fragment size; Removed run number from generic header Included Run Number in list of Full Event Specific elements; Removed Sub-detector specific header elements as this is just a straight copy of what is in the Full Event Specific elements; Removed ROB specific header elements. In previous versions these elements were copied from a ROD fragment, i.e. ROS does not receive this information in-dependently; Removed LVL1 result specific header elements; Change description of Event Filter output to reflect removal of Offset elements; Modified description of use of status elements. New usage reduces overall frag-ment size; Release to sub-set of community for initial immediate feed-back
3	0	11 Apr. ‘05	Added a ‘Test’ run type, see page 16; Release for comment; Removed LVL1 Result Start of Header Marker; Updated Appendix A to give bit definition of control words; Removed some typing errors; Clarified in section 2.3 that 1 ROB fragment contains only 1 ROD fragment
		01 June ‘05	Clarified that for a ROD fragment the source ID contains the ROL number and NOT ROD module ID; Released for information in EDMS
3	1	27 Nov. ‘06	Updated the Full Event header Specific elements to take into account Luminosity block and streaming proposals, section 5.9.1; Run number now defined as a 31-bit integer; Added Appendix B: pending issues; Added reference to document defining Status elements in ROS and ROB frag-ments, section 5.8; Added definition of the Status Elements in the Full Event Header, section 5.8

		1 Dec. ‘06	Removed some typing errors
		14 Dec. ‘06	Added the run type to the Event Specific header, section 5.8 Level 2 and Event Filter info words in Event specific header redefined, section 5.8, i.e. replace previous definitions of trigger info and type; Removed ambiguity on Module ID in ROD fragments; Format of bunch crossing time added; Changed major version number to 3.1
4	0	23 Jan. ‘08	Format simplification removing Sub Detector and ROS headers; Introduced ROB trailer (check-sum) in consultancy with the ROS working group; Clarified the meaning of specific status words in both ROB and Full Event fragments; Removed the Detector Mask field in the Full Event Header; Updated References; Disentangled major versions between ROD fragments and the remaining fragment types; Changed major version number to 4.0; Removed Appendix on to-dos. Included Appendix on the ROB trailer;
		24 Jan. ‘08	Included comments from Hans Peter
		25 Jan. ‘08	Included reference on ROB check-sum algorithm Explain Full Event
		29 Jan. ‘08	Better text to some event fragments, emphasized wording Page numbers were missing ROBIN status: wording changed; removed exclamation marks; bit 24 is always zero Removed Appendix on Framing Introduced check-sums as part of the standard fragment; Updated Appendix on check-sums to include information on Adler-32 Changed Pixel Detector Source Identifiers
		04 Feb. ‘08	Incorporated number of modifications proposed by FJW Reserved field in Source Identifiers becomes “optional” Clarification on the interpretation of Status Elements
		12 Feb. ‘08	Clarifications proposed by MJ Introduction of the “Simulation” flag in the Run Type table Introduction of the Forward detectors at the Subdetector Identifier table Clarify that, if a ROS builds a Full Event object, some fields may not be properly initialized
		02 Sep. '08	Fix in forward detector Subdetector Identifier table by HPB
		14 Oct. '08	Specify HLT protocol for decoding the status words in HLT results
4	0c	09 Feb. '09	We now refer to the ROBIN status words at their Wiki page instead of redefining them here Added “Laser Crate” (0x50) subdetector identifier

Table 1: Document change record

1 Introduction

1.1 Purpose of the document

This document describes the raw event format and its implementation in the ATLAS Trigger and DAQ.

1.2 Overview

In Section 2 the requirements, function, purpose and a high-level description of the event format is given. In Section 3 a detailed description is given. Section 4 presents a description of the format of a fragment sent by a ROD over a ROL. In Section 5 the implementation of the event format is described. Appendix A covers details on the check-sum trailer (algorithms and reference).

1.3 Boundaries

This document relates to the format of data into and out of the: RODs, Read-Out Sub-system (ROS), Data Collection Sub-system, the LVL2 Selection and Event Filter Subsystems of the Higher Level Triggers (HLT). The framing information, necessary to ensure the correct transmission of data between applications, e.g. ROD-to-ROB, is technology specific and therefore not part of the event format.

1.4 Definitions, acronyms and abbreviations

See reference [1].

2 General Description

2.1 Requirements

This sub-section lists a set of requirements on the various components of the event format. The categories of requirements follow the guidelines given in [2]. Requirements containing the word shall are mandatory. Those containing the word should are strongly recommended, justification is needed if they are not followed.

The event format shall fulfil the following requirements:

1. The event format *shall* allow the size of an event to increase or decrease depending on the specific data taking configuration;
2. There *shall* be no minimum or maximum event data size implied by the format;
3. The event format *should* provide information redundancy to allow self consistency checks of the event to be made;

4. The event formatting information *shall* not exceed 20% of the typical full ATLAS event data size;
5. The event format *should* be modular;
6. The basic unit *should* be a fragment. Fragments are: parts of an event coming from a ROD or ROB or the (Full) Event itself;

   *Please note that the term **Full Event** refers to the fragment with a preceding event header and containing **ROB** fragments. The number of fragments held internally may not represent the whole of the detector data for a particular Level-1/Global identifier.*
7. The fragments *should* have identical structure;
8. The event format *shall* facilitate the identification of fragments;
9. The event format *shall* provide an event header;
10. The event format *shall* provide the event identifier and trigger type within the full event header;
11. The event format *shall* provide a means of identifying whether the event has been corrupted during transmission within the Data Flow, e.g. DMA time-out, truncation etc;
12. The event format *shall* provide a means of identifying whether the event has been corrupted due to hardware problems, e.g. a bit error.

2.2 Function and purpose

The event format defines the structure of the data at various stages within the HLT and DAQ, and allows elements of the Data Flow and HLT processing tasks to access the data without resorting to the use of other resources, e.g. data bases. In addition, it defines additional data that is added to the detector data, by elements of the TDAQ, allowing processing tasks to quickly identify the type and origin of each event.

2.3 General format

The general format of a Full Event is shown in Figure 1. As can be seen it is built from fragments (see Requirement 6. in Section 2.1). A Full Event is an aggregation of ROB fragments. Each of the latter map on to a **single** ROD fragment. Each fragment type, except the ROD fragment, has a header which contains all the event formatting information need to decode it. Besides the generic header, ROB and Full Event fragments may contain a single 32-bit word trailer with a check-sum of its contents (see Section 6). For ROD and ROB fragments, hardware considerations have led to the combination of a header and a trailer, however, the general principles are similar and it is the combination of the header and trailer which provide the event formatting information required to decode it. Details of ROD fragments are given in Section 4.

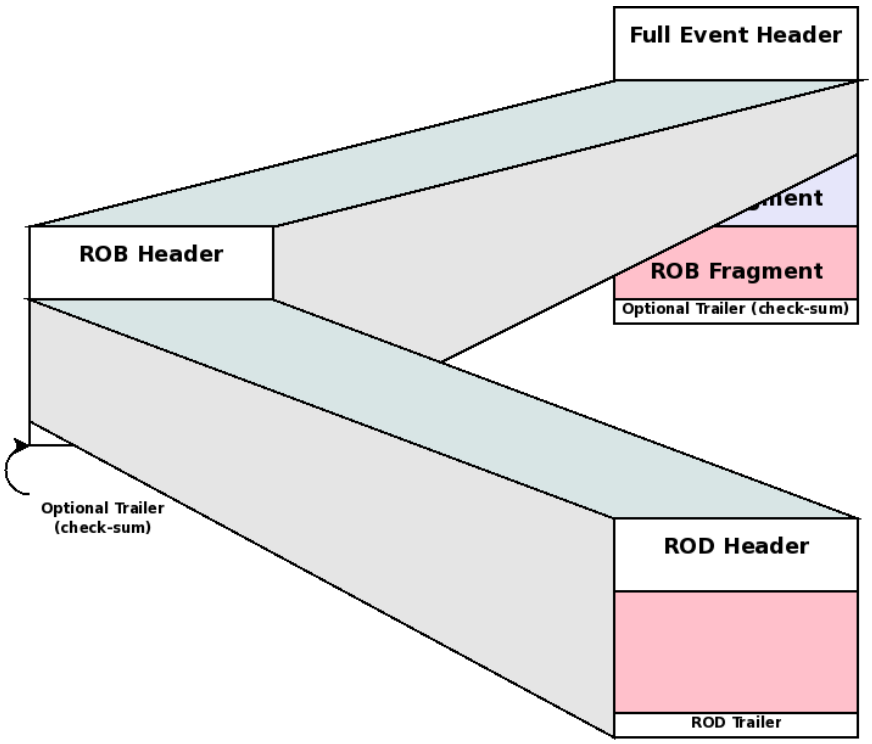


Figure 1: The general event format.

The class diagram of the raw event format is shown in Figure 2. Referring to the latter, it can be seen that Full Events and ROB fragments are types of Fragments, which are characterized by a common Generic Header. Full Event fragments contain any number of ROB fragments each of which contains a single ROD fragment. Full Event fragments extend the Fragment type header with specific fields, as it will be shown in Section 3.

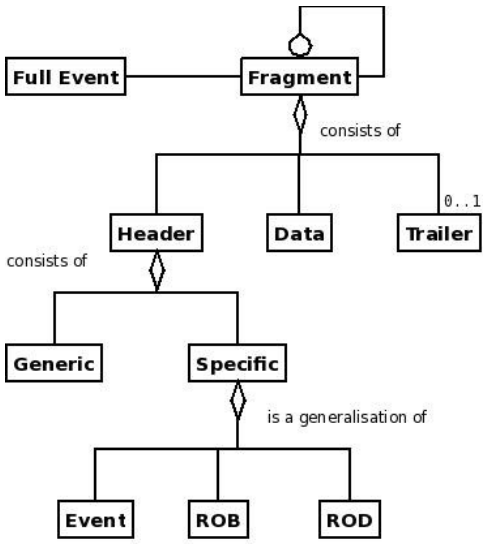


Figure 2: The class diagram of the raw event format.

As can be seen from Figure 2 the proposed raw event format is modular and based on event fragments (see Section 2.1). All event fragments have the same structure, ex-

cept the ROD fragment due to identified implementation issues. This fulfils Requirement 7. (see Section 2.1).

3 Header formats

3.1 The Header

Start of Header Marker	**Generic**
Total Fragment Size	
Total Header Size	
Format Version Number	
Source Identifier	
Number of Status Elements (N)	
Status Element[0]	
...	
Status Element[N-1]	
Check Sum Type	
Specific Header[0]	**Specific**
...	
Specific Header[M]	

Table 2: The (generic) fragment header.

The Header type is an aggregation of Generic and Specific parts, see Table 2. The Generic part is the same for Full Event and ROB fragments and slightly different for ROD fragments (see Section 4). The Specific part allows fragment specific information to be included in the header.

3.1.1 The Generic component

The Generic component consists of the following elements:

1. *Start of header marker*: This marker indicates the start of a fragment header and is itself part of the header. Hence, it is the first word of a fragment. The value of this element will be unique for each type of fragment, but the structure shall be identical. The structure will allow the endianess of the fragment header to be determined;
2. *Total fragment size*: This element indicates the total size of the fragment, including the Header;
3. *Header size*: The element indicates the total size of the Header;
4. *Format version number*: This element gives the format version of the fragment;
5. *Source identifier*: This element identifies the origin of the fragment. It consists of a sub detector ID, and Module ID. The combination of these fields should allow

the Source identifier to be unique across the whole of Atlas. The Module ID refers to the module which builds and adds the header to the event fragment;

6. *Number of status elements*: The value of this element is the number of status elements in the Header;
7. *Status element*: This element contains information about the status of the data within the fragment. The structure of this element is specific to the module which builds the header;
8. *Check Sum Type*: This element indicates if this fragment contain a check-sum attached to its trailer, as a single 32-bit word, depending on this value. This field can currently take the following values:

Value	***Description***
0x0	No check-sum is present (no trailer)
0x1	A CRC-16/CCITT check is available
0x2	An Adler-32 check is available

A value of 0x0 indicates this fragment has no trailer and therefore no check-sum against its payload. A value different from zero indicates a check-sum is available. The algorithm applied for its calculation is defined by the table above. Check on Appendix A for details and references.

3.1.2 The specific component

Following the Generic component of the header there is a fragment Specific component consisting of zero or more words, depending on the fragment type. See Section 5.9 for details.

4 ROD data format

The definition of the format of the data transferred between the ROD and ROB must take into account factors such as: the data is formatted in hardware and not necessarily by programmable devices; the information within the header may influence component cost and ROD performance; the differences in ROD designs.

To accommodate the differences in the ROD designs the data transferred from a ROD to a ROB should have both a Header and a Trailer as shown in Figure 3.

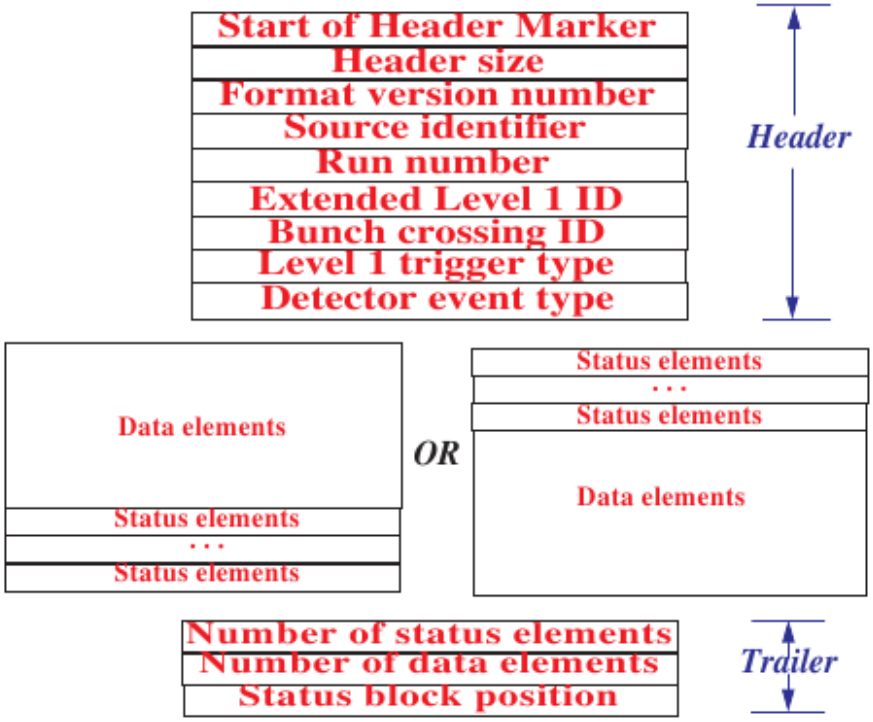


Figure 3: The ROD fragment format.

The Trailer contains the Number of data elements, Number of Status elements and the status block position. Some detector groups have voiced a preference for having the Status elements proceeding the Data elements. Instead of imposing an order, an additional element, Status block position, has been added to the trailer. The value of this element defines the relative order of the Data and Status elements. A value of zero indicates that the status block precedes the data block and a value of one indicates that the status block follows the data block. These two cases are shown in Figure 3 for reasons of clarity. The Data and Status elements are 32-bit integers.

The header is derived from that presented in Section 3.1 and, with the exception of the Source Identifier, the elements have the same meaning. For a ROD fragment the value contained in bytes 0 and 1 of the Source Identifier is the ROL number, unique to each ROL. Note, the value of the Start of Header Marker also identifies the byte order of the ROD fragment Data and Status elements.

Within the ROD fragment header five additional elements are explicitly defined, these are:

1. *Run Number*: An element whose value is unique during the lifetime of the experiment (see Section 5.4);
2. *Extended Level 1 ID*: The Extended L1ID [3] formed by the 24-bit L1ID generated in the TTCrx and the 8-bit ECRID implemented in the ROD;
3. *Bunch Crossing ID*: The 12-bit bunch crossing identifier generated in the TTCrx;
4. *Level 1 Trigger Type*: The 8-bit word generated by the Central Trigger Processor or LTP and transmitted by the TTC system [4]. The remaining 24-bits are unused and set to zero.

   Note, a value of zero indicates a ROL Test Block as described in [5];

5. *Detector event type*: This element allows additional information to be supplied on the type of event, particularly in the case of calibration events. It allows the detectors to specify the exact type of calibration event that they have generated.

The **first** status word shall indicate the global status of the fragment. A non-zero value of this element indicates that the data payload of the fragment is corrupted, e.g. missing data and or bit errors, see Section 5.8.

5 Implementation

This section presents an implementation of the event format described in the previous sections. It defines the Start of Header Markers, the Fragment IDs, the sub-detector IDs and the elements specific to the different types of fragments. This implementation is for 32-bit machines and demands that the Generic Header, ROD Header and Trailer are aligned on four byte boundaries. All header and trailer elements are 32-bit integers. Note, future evolution of the event format may demand eight byte alignment.

In this implementation: the ROD and ROB header are built by the ROD and ROB respectively; the Event Header is built by the SFI. The implementation of the Event Format does not impose a specific order of the fragments. It follows that the user should not rely on any particular ordering information to be constant among different events: *e.g.* the first ROB fragment on an event may have source identifier set to Pixel Disk, module id. equals to 34, followed by a ROB fragment with source identifier set to LArg EM C-Side, module id. 5. For the next event, the sequence may be completely different with respect to detector identifiers or specific module identifiers.

The following points have also been taken into account:

- *Floating point types* are not used in this implementation as they are not portable;
- *Byte ordering*: The endianness of the ROD fragment is defined in [6].

  The fragments generated by TDAQ components shall be little endian, reflecting the fact that all processing nodes house little endian processors. The implementation of the Start of Header Marker allows the endianness of the fragment to be verified, see Section 5.1.
- *Alignment*: The implementation demands that all headers are aligned on 4-byte boundaries.

5.1 Start of Header Markers

Each fragment header begins with a Start of Header Marker. These markers fulfil Requirements 7, 8 and 9 as described in Section 2.1. The markers at each level of the event format are given in Table 3.

The asymmetry in the value of the Start of Header Marker allows for the byte ordering used in the fragment Header to be identified. Note, for the ROD fragment it refers to the byte order of the ROD fragment as a whole.

Fragment Type	Header Marker
ROD	`0xee1234ee`
ROB	`0xdd1234dd`
Full Event	`0xaa1234aa`

Table 3: Start of Header Markers.

5.2 Source Identifiers

The structure of the Source identifier, as shown below, consists of three byte fields. The combination of these three fields allows the Source identifier to be unique across all sub-detectors. The values that the Sub-detector identifier may have are defined in Section 5.3. The value of the Module ID for a ROD fragment is the ROL number and is unique to each ROL. For other fragments, the value that may be assigned to the Module ID is free to be defined by the system or sub-system implementers concerned.

Byte	*3*	*2*	*1*	*0*
	Optional (= 0x0)	**Sub-Detector ID**	**Module ID**	

Byte 3 is optional and should be initialised to a value of zero. The value of this field *may* be used by implementers of TDAQ processors to carry extra information about hardware connectivity, for **debugging purposes**. An example use-case for this field happens in the ROS: its identifier is placed in this reserved field for the ROB fragment it produces and can be used for debugging hardware connection problems.

5.3 Sub-Detector IDs

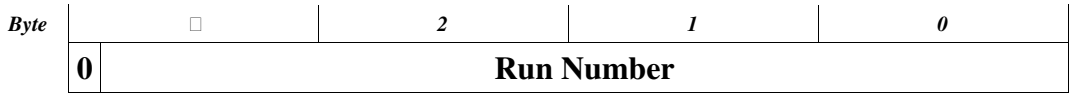
The values that the Sub-detector ID field may have are given in Table 4. Values not listed in this table are considered illegal.

Detector		ID	Detector		ID
Full Event		0x00	***Muon***	*MDT Barrel A Side*	0x61
Pixel	*Barrel*	0x11		*MDT Barrel C Side*	0x62
	Disk	0x12		*MDT Endcap A Side*	0x63
	B-layer	0x13		*MDT Endcap C Side*	0x64
SCT	*Barrel A Side*	0x21		*RPC Barrel A Side*	0x65
	Barrel C Side	0x22		*RPC Barrel C Side*	0x66
	Endcap A Side	0x23		*TGC Endcap A Side*	0x67
	Endcap C Side	0x24		*TCG Endcap C Side*	0x68
TRT	*Barrel A Side*	0x31		*CSC Endcap A Side*	0x69
	Barrel C Side	0x32		*CSC Endcap C Side*	0x6a
	Endcap A Side	0x33	***TDAQ***	*Calorimeter Preprocessor*	0x71
	Endcap C Side	0x34		*Calo Cluster processor DAQ*	0x72
LArg	*EMB A Side*	0x41		*Calorimeter Cluster processor RoI*	0x73
	EMB C Side	0x42		*Calo Jet/Energy processor DAQ*	0x74
	EMEC A Side	0x43		*Calo Jet/Energy processor RoI*	0x75
	EMEC C Side	0x44		*Muon CTP Interface (MuCTPI)*	0x76
	HEC A Side	0x45		*CTP*	0x77
	HEC C Side	0x46		*L2SV*	0x78
	FCAL A Side	0x47		*SFI*	0x79
	FCAL C Side	0x48		*SFO*	0x7a
TileCal	*Laser Crate*	0x50		*Level-2*	0x7b
	Barrel A Side	0x51		*Event Filter*	0x7c
	Barrel C Side	0x52	Forward	*BCM*	0x81
	Extended A Side	0x53		*Lucid*	0x82
	Extended C Side	0x54		*ZDC*	0x83
				Alpha	0x84

Table 4: Sub-detector Ids.

5.4 Run number

This element is 32-bits. The run number is a 31-bit integer and the highest bit is zero.



5.5 Total fragment and header size

These elements are each 32-bit integers and their values give the total size of the fragment and the size of the fragment header in units of 32-bit integers.

5.6 Format version number

This element consists of two 16-bit fields, as shown below. The combined value of these fields identifies the fragment format version. The Major version number shall be the same for all fragments in the event that possess a Generic Header. This assertion excludes ROD fragments, which may ship with a different (older) version number. The Minor version number has a value dependent on the fragment type and will be used to identify the format of the specific part of the fragment header and in a ROD fragment the format of the sub-detector Data.

The implementation described in this document defines the Format Version Number to be 4.0-0.0 (0x04000000), i.e. Major version number is **4.0** and the Minor version number is 0.0. For ROD fragments, the Major version number is **3.1** and the Minor version is free to be chosen by the specific Sub Detector groups.

Byte	*3*	*2*	*1*	*0*
	Major version number		**Minor version number**	

5.7 Number of status elements

A value of zero indicates that there are no subsequent Status elements and therefore there are no known errors associated to the fragment.

5.8 Status elements

This element is a 32-bit integer. The **first** Status element shall be divided into two 2-byte fields labelled Generic and Specific, see below. The values and error conditions indicated by the Generic field are the same for all fragments, while the values and error conditions indicated by the Specific field have meanings specific to the fragment. A non-zero value of this element indicates that the event fragment has a problem, *e.g.* truncated. The information conveyed by the status element only refers to the fragment of which it is an element.

The remaining status elements **following** the first word of a fragment may have different formatting, to be defined by the implementers of the specific software or hardware that creates or manipulates these fragments.

Byte	*3*	*2*	*1*	*0*
	Specific		**Generic**	

The currently defined values and meanings of the Generic field of the first Status element are given in Table 5.

Generic Field Value	*Description*
0 (0x00)	Unclassified
1 (0x01)	An internal check of the BCID has failed.
2 (0x02)	An internal check of the EL1ID has failed.
4 (0x04)	A time out in one of the modules has occurred. The fragment may be incomplete.
8 (0x08)	Data may be incorrect. Further explanation in the Specific filed.
16 (0x0f)	An overflow in one of the internal buffers has occurred. The fragment may be incomplete.

Table 5: Values and meaning for the Generic field of the first status element.

The analysis of the **first** status word should follow these principles:

1. If there are **no** status words in a fragment, then there are **no** known problems with that fragment;
2. If there **are** status words associated with a fragment:
    1. If the first status word is **all** zeroes (**both** generic **and** specific parts together) then there is **no** know problem with the fragment;
    2. If either the generic **or** specific parts of the first status words are **not** zero, a problem might have occurred in which case the user is expected to understand the contents of the tables in this section.

5.8.1 ROB specific status

The meaning and values of the Specific field of the Status elements in the ROB header are given Reference [7].

5.8.2 HLT results the ROB specific status

Because HLT results (L2 and EF) are wrapped around common ROB fragments, the specific status bits defined in Reference [7] may also apply, where relevant, to this kind of fragment. In particular, the following protocol in decoding the HLT result ROB status should be used:

1. If an error is signalled in the generic part of the first status word of an HLT generated ROB (L2 or EF results), details are specified in the specific part of this word;
2. In case 1 holds, and all bits in the specific part are not set, the reported issue was detected by the HLT framework and more details **may** be available in the remaining status words attached to this fragment header;
3. In case 1 holds, and any bit in the specific part is set, the reported issue was detected by the Dataflow framework. The existing data in the result payload is either empty or dummy and cannot be interpreted by the HLT framework.

While decoding the HLT result, the HLT or Offline frameworks can use further information provided in the specific part of the first status word in the Full Event header to complement on the information provided by the HLT result fragment itself.

5.8.3 Full Event specific status

The meaning and values of the Specific field of the Status elements in the Full Event header are given in Table 6. For each of these values the Generic field should assume

the value of 0x8 (*Data may be incorrect*). This table also describes the agents in the Data Flow system that can set these fields.

Bit	Meaning	Agent
16	L2_PROCESSING_TIMEOUT	*L2SV*
17	L2PU_PROCESSING_TIMEOUT	*L2PU*
18	SFI_DUPLICATION_WARN	*SFI*
19	DFM_DUPLICATION_WARN	*DFM*
20	L2PSC_PROBLEM (*)	*L2PU*
21	*Reserved (=0x0)*	
22	*Reserved (=0x0)*	
23	*Reserved (=0x0)*	
24	EF_PROCESSING_TIMEOUT	*EFD*
25	PT_PROCESSING_TIMEOUT	*PT*
26	SFO_DUPLICATION_WARN	*EFD*
27	EFD_RECOVERED_EVENT	*EFD*
28	EFPSC_PROBLEM (*)	*PT*
29	EFD_FORCED_ACCEPT	*EFD*
30	*Reserved (=0x0)*	
31	*Reserved (=0x0)*	

Table 6: Values and description of the Specific field in the Full Event status element.

(*) Note: In the occurrence of a PSC Problem, indicated by flags on bits *20* (for Level-2) or *28* (for Event Filter), the PSC may use more status words following the first one to indicate the exact cause of the problem. This protocol remains private to agents that can effectively make use of this information – all being either on the HLT or Off-line software domains. Therefore, the meaning of the words following the first status word is not detailed in this document.

5.9 Fragment specific elements

5.9.1 Full Event specific elements

The Full Event specific elements are defined in Table 7. Each element in this table is padded to form a 32-bit word. The table also presents the required order of the specific elements and the definition is invariant with respect to the run type.

Event Header words	*Definition*
Bunch crossing time (seconds)†	32-bit integer
Bunch crossing time (nanoseconds)†	32-bit integer
Global event ID†	32-bit integer
Run type	32-bit integer
Run number	32-bit integer
Luminosity block number†	**16**-bit integer
Extended Level 1 ID	32-bit integer
BCID	**12**-bit integer
Level 1 trigger type	**8**-bit integer
# Level 1 trigger info words	32-bit integer
1st Level 1 trigger info word	32-bit integer
Nth Level 1 trigger info word	32-bit integer
# Level 2 trigger info words	32-bit integer
1st Level 2 trigger info word	32-bit integer
Nth Level 2 trigger info word	32-bit integer
# Event Filter info words	32-bit integer
1st Event Filter info word	32-bit integer
Nth Event Filter info word	32-bit integer
# Stream Tag words	32-bit integer
1st Stream Tag word	32-bit integer
Nth Stream Tag word	32-bit integer

Table 7: Fragment specific header for the Full Event.

- *Bunch crossing time*: This element is the bunch crossing time. It is two 32-bit integers encoding the GPS time of the bunch crossing as recorded by the Central Trigger Processor and is the number of seconds and nanoseconds since 1st January 1970 (in two separated fields). The Full Event fragment is built by the SFI, therefore the value of this element is copied from the Level 1 CTP fragment into these fields;
- *Global event ID*: The value of this 32-bit integer will be provided by the DFM component of the Event Building subsystem. The value will be unique within a run;
- *Run type*: This element is 32-bits. A **preliminary** enumeration of Run Type is given in Table 8.

Run Type	**Value**
Physics	0x00000000
Calibration	0x00000001
Cosmics	0x00000002
Test	0x0000000f
Simulation	0x80000000

*Table 8: Enumeration of Run Type. The "simulation" flag (last, MSB bit), may be set **together** with any of the previous entries to indicate the data origin (Monte Carlo).*

- *Run number*: This element is 31-bits, the most significant bit is unused;
- *Luminosity block number*: This element is 16-bits;
- *Extended Level 1 ID*: The extended LVL1 ID [3] formed by the 24-bit L1ID generated in the TTCrx and the 8-bit ECRID implemented in the ROD;
- *BCID*: The 12-bit bunch crossing identifier generated in the TTCrx;
- *Level 1 trigger type*: An 8-bit word as generated by the Central Trigger Processor and transmitted by the TTC system [4]. The remaining 24-bits are un-used and set to zero;
- *# Level 1 trigger info words*: The number of Level 1 trigger info words, **excluding** this one;
- *Level 1 trigger info:* A number (given by the previous element) of 32-bit words summarising the Level 1 trigger chain. The exact details of these words are not yet defined;
- *# Level 2 trigger info words*: The number of Level 2 trigger info words, **excluding** this one;
- *Level 2 trigger info*: A number (given by the previous element) of 32-bit words summarising the Level 2 trigger chain. The exact details of these words are not yet defined;
- *# Event Filter info words*: The number of Event Filter info words, **excluding** this one;
- *Event Filter info*: A number (given by the previous element) of 32-bit words summarising the Event Filter chain. The exact details of these words are not yet defined;
- *# Stream tag words*: This is the number of encoded stream tag words, **excluding** this one;
- *Stream tag*: This element is a C-string identifying to which data stream or data streams the event has been assigned. It is a null-terminated C-String. The composition of each stream tag is the following
  - *name*: defines the name of the tag. It is a free string;
  - *type*: defines the type of the tag. It is a free string;
  - *obeys_lumiblock*: defines if the event obeys the lumiblock boundaries or not. It is a boolean value.

Whenever detector or simulated data is recorded, a Full Event header shall be used to wrap the various ROB fragments. In case Full Event fragments are produced directly from a ROS dump (*e.g.* when it records detector data for debugging purposes), fields marked with the symbol "†" at Table 7 will **not** be properly initialized due to the nature of this component. In this case:

- The *Run Type* field will be set to "Test" (`0xf`);
- The *Bunch Crossing* time entries should be both set to zero (`0x0`).

5.9.2 ROB specific header

No fragment specific elements are currently defined.

5.10 Event Filter output

The input to the Event Filter is a Full Event fragment. The output of the Event Filter shall be the same fragment with an additional ROB fragment appended. The value of the Sub-detector identifier in the Source identifier of this additional fragment, shall be equal to that of the Event Filter.

5.11 ROD header and trailer

The initial implementation of the ROD header and trailer has been given in Section 4. These elements, including the Data and Status elements, are 32-bit integers, e.g. The *Level 1 Trigger type* is an 8-bit value, therefore the remaining 24-bits are unused.

6 Optional Check-sum

6.1 The ROBIN/ROD check-sum

The ROBIN hardware is able to perform a check-sum on its payload, the **whole ROD fragment**. It can, *optionally*, ship this check-sum as a single 32-bit value just after the ROB fragment itself, forming a single word trailer. The chosen algorithm is a CRC-16/CCITT sum [8], which can be easily executed in hardware.

Because it is a 16-bit CRC sum, it is executed over the most significant 16-bit word separately from the less significant part. Both CRC sums are then stored in the ROB trailer, at equivalent places.

6.2 The Full Event fragment check-sum

The Full Event fragment may *optionally* carry a check-sum trailer. This check-sum maybe calculated using either CRC-16/CCITT or Adler-32 [9]. The later algorithm requires less computational power, being better adapted for calculations within HLT or Off-line processors being therefore the recommended one in those cases.

Following the model in the ROBIN check-sum, the check-sum sitting in the Full Event Fragment trailer applies only to this fragment's payload, i.e., the ROB fragments it contains. In the case an Adler-32 check-sum is used, this technique allows for a “rolling” check-sum calculation to be deployed while the event is being constructed or extended (*e.g.* by the Event Filter). Event modifications (*e.g.* the suppression of parts of an event) would still require the check-sum to be recomputed.

The recommended implementation for Adler-32 sits in zlib [10].

References

[1] ATLAS TDAQ, ATLAS High-Level Trigger Data Acquisition and Controls Technical Design Report, CERN, Appendix B, CERN/LHCC/2003-022 (2003)

[2] C. Mazza, Software Engineering Standards, Prentice Hall, ISBN: 0-13-106568-8

[3] R. Spiwoks, Presentation given to the Front-end Electronics Co-ordination on 27/February/2002,

[4] Level-1 Trigger Group, Definition of the trigger-type word, CERN, EDMS: ATL-DA-0022

[5] R. McLaren, ROL Test Block, CERN, https://edms.cern.ch/document/439294/1

[6] R. McLaren, ATLAS Read-Out Drivers: Endianness, CERN, EDMS, ATC-TD-EC-0001

[7] Markus Joos, ROBIN Specific Status, CERN, https://twiki.cern.ch/twiki/bin/view/Atlas/ROBINFragmentErrors

[8] Cyclic redundancy check, http://en.wikipedia.org/wiki/Cyclic_redundancy_check

[9] Adler-32, http://en.wikipedia.org/wiki/Adler-32

[10] Zlib, http://www.zlib.net/